RAPPORT DE STAGE

Développement d'une maquette pour l'étude de la cybersécurité appliquée à l'IoT

Réalisé par
DJENADOU Rayane

Étudiant en *Master 1 TDSI - Parcours Objets Connectés*

Effectué au sein de **l'Université de Poitiers - Département EEA**

Du **19/05/2025** au **18/07/2025**

Enseignant référent :
PERRINE Clency

Tuteur de stage :
BOEGLEN Hervé

Année universitaire 2024–2025

Remerciements

Je tiens à exprimer ma gratitude à toutes les personnes qui ont contribué, de près ou de loin, au bon déroulement de ce stage.

Je remercie tout particulièrement Marc Parenthöen pour m'avoir accueilli en stage et pour l'opportunité enrichissante qu'il m'a offerte.

Je souhaite également adresser mes sincères remerciements à Hervé Boeglen, mon tuteur de stage, ainsi qu'à Noël Richard, pour la qualité de leur encadrement, leur expertise et la confiance qu'ils m'ont accordée tout au long de mon travail au sein de l'Université de Poitiers.

Enfin, je remercie chaleureusement mon collègue Maxime Moret pour son implication, sa disponibilité et l'excellente collaboration tout au long de ce projet.

Table des matières

Table des figures

1 Introduction

1.1 Contexte du stage

Ce stage a pour objectif de réaliser une maquette destinée à l'étude de la cybersécurité dans le domaine de l'Internet des Objets (IoT). Cette maquette a également vocation à être présentée lors de salons étudiants ou d'événements de formation, afin de sensibiliser le public aux enjeux actuels de la sécurité informatique dans les systèmes connectés, et de faire découvrir les domaines de l'IoT, de la cybersécurité, des systèmes embarqués et des technologies associées.

La maquette doit être à la fois pédagogique, ludique et visuellement accessible. Elle a été pensée pour permettre une interaction simple, tout en illustrant de façon concrète des scénarios réalistes d'attaque et de défense dans un environnement connecté.

Ce projet vise également à mettre en valeur les compétences développées au sein du Master Objets Connectés de l'Université de Poitiers. À travers cette maquette, il s'agit de démontrer de manière pratique les savoir-faire mobilisés en matière de développement embarqué, communication réseau et cybersécurité, tout en restant compréhensible pour un public non expert.

L'IoT étant en pleine expansion, les enjeux liés à sa sécurisation sont devenus majeurs. Ces dispositifs, de plus en plus présents dans notre quotidien, exposent de nouveaux risques (écoute réseau, intrusion, déni de service, etc.). Il est donc essentiel d'apporter une réponse pédagogique et technique à ces problématiques.

Ce rapport présente les différentes étapes de ce projet, de l'analyse des besoins à la réalisation concrète, en passant par la conception, le développement, la sécurisation, et l'évaluation de la solution mise en place.

1.2 Présentation de la structure d'accueil

L'Université de Poitiers, fondée en 1431, est l'une des plus anciennes universités d'Europe. Elle propose un large éventail de formations dans les domaines scientifiques, techniques, littéraires et juridiques, et elle est reconnue pour la qualité de sa recherche et de son enseignement supérieur.

Le stage a été réalisé au sein du Campus du Futuroscope, un pôle universitaire et technologique moderne rattaché à l'Université de Poitiers. Situé à Chasseneuil-du-Poitou, ce campus accueille de nombreuses formations orientées vers les sciences et technologies de l'information, les systèmes embarqués, l'intelligence artificielle et la cybersécurité. Il bénéficie d'un environnement propice à l'innovation, avec la proximité d'entreprises, de laboratoires de recherche et d'infrastructures techniques avancées.

Durant ce stage, les travaux ont été menés principalement dans les salles 0N06 et 0N16, situées au sein du bâtiment SP2MI. Ces espaces sont dédiés aux projets technologiques et disposent des équipements nécessaires pour le prototypage, la programmation de systèmes embarqués et les expérimentations réseau. Ce cadre dynamique a permis de concevoir, développer et tester l'ensemble de la maquette dans des conditions proches d'un environnement réel.

Encadré par des enseignants-chercheurs expérimentés, le stage s'est inscrit dans une logique collaborative et pédagogique. Il a permis de développer une maquette interactive destinée à illustrer les enjeux de cybersécurité dans l'Internet des Objets (IoT), tout en valorisant les compétences acquises dans le cadre du Master.

2 Analyse des besoins et cahier des charges

2.1 Objectifs fonctionnels et pédagogiques de la maquette

La maquette développée dans le cadre de ce stage vise à remplir un double objectif : démontrer des fonctions concrètes liées à un système IoT, et servir de support pédagogique pour la sensibilisation à la cybersécurité.

Sur le plan fonctionnel, la maquette doit :

- Mettre en œuvre un circuit à billes physique composé de capteurs, et de microcontrôleurs répartis sur plusieurs zones ;
- Utiliser six microcontrôleurs qui communiquent avec un tableau de bord distant via des protocoles sans fil ;
- Collecter et transmettre en temps réel des données capteurs sur un tableau de bord interactif, permettant un suivi visuel en direct du fonctionnement du circuit ;
- Déclencher des actions physiques (activation de vérins, LEDs, changement de trajectoire) en fonction des événements détectés ou des commandes reçues ;
- Simuler des scénarios d'attaque réseau, observables via un microcontrôleur sniffer intégré ;
- Intégrer des éléments visuels et sonores (lumières LED, effets audio) pour créer une ambiance ludique et immersive, proche d'une borne d'arcade.

Sur le plan pédagogique, la maquette doit permettre :

- D'expliquer concrètement et visuellement le fonctionnement d'un système IoT ;
- D'illustrer les impacts réels de failles de cybersécurité dans un système connecté (modification de comportement, fausse donnée, contrôle externe) ;
- De montrer comment protéger un système IoT contre les attaques, en appliquant des mesures simples mais efficaces comme le chiffrement des données et l'authentification.
- De valoriser les compétences acquises en Master Objets Connectés (développement embarqué, communication sans fil, cybersécurité, intégration multimédia) ;
- De proposer une démonstration immersive, ludique et engageante à destination d'un public large, dans un format attractif type “arcade éducative”.

2.2 Cahier des charges et contraintes techniques

Aucun cahier des charges formel ne nous ayant été transmis, nous avons défini un ensemble de contraintes techniques et pratiques à respecter, en cohérence avec les objectifs fonctionnels décrits précédemment.

La maquette devait être pensée pour des usages pédagogiques et des démonstrations publiques (salons étudiants, événements grand public). Cela impliquait un certain nombre de contraintes concrètes, notamment :

Transportabilité : l'ensemble devait tenir dans le coffre d'une voiture, et être facile à transporter.

Autonomie : le système devait fonctionner dès la mise sous tension, sans configuration préalable ni intervention technique. Il ne devait nécessiter aucun ajustement manuel.

Indépendance réseau : la maquette devait fonctionner en réseau local autonome, sans connexion Internet ni infrastructure externe. Toutes les communications devaient être gérées localement, entre les microcontrôleurs et l'interface de supervision.

Immersion : des effets lumineux et sonores devaient être intégrés afin de rendre les démonstrations plus attractives, ludiques et intéressantes à observer. L'ensemble devait évoquer une expérience proche de celle d'une borne d'arcade, avec des animations dynamiques et visuellement engageantes, renforçant l'aspect interactif et pédagogique de la maquette.

Aspect cybersécurité : le projet intégrait des scénarios de cybersécurité (comme l'interception de données ou l'usurpation d'identité), mais dans une optique pédagogique. Pour cela, certaines hypothèses ont volontairement été simplifiées : absence de chiffrement des communications, authentification non implémentée, etc. L'objectif n'était pas de reproduire un environnement réaliste, mais de faciliter la compréhension des enjeux et des mécanismes de sécurité dans les systèmes IoT.

2.3 Analyse préliminaire du projet

2.3.1 Architecture détaillée de la maquette

L'architecture détaillée du projet repose sur une organisation modulaire et articulée autour de plusieurs microcontrôleurs (STM32, nRF7002 DK), chacun dédié à une zone fonctionnelle précise de la maquette. Ces microcontrôleurs sont reliés à des capteurs et actionneurs locaux, et interconnectés via différents protocoles de communication (BLE, Wi-Fi, MQTT, HTTP) avec un serveur central (Raspberry Pi) hébergeant l'interface de supervision ThingsBoard ainsi que les services associés (base de données, broker MQTT, passerelle BLE vers MQTT).

Chaque partie de la maquette (zone du circuit à billes) est contrôlée par un ou plusieurs microcontrôleurs, qui assurent à la fois la détection des événements (passage d'une bille, reconnaissance de couleur...) et le déclenchement de réactions physiques (activation d'un vérin, d'un servomoteur, éclairage, etc.).

Les informations sont ensuite centralisées sur un Raspberry Pi, qui héberge les éléments suivants :

- un broker MQTT (Mosquitto) servant de point de transit pour tous les messages du système,
- une base de données PostgreSQL,
- une interface ThingsBoard, servant de tableau de bord interactif,
- une passerelle BLE vers MQTT permettant aux STM32 BLE de communiquer avec le système principal.

Le schéma ci-dessous présente les interactions fonctionnelles détaillées du système :

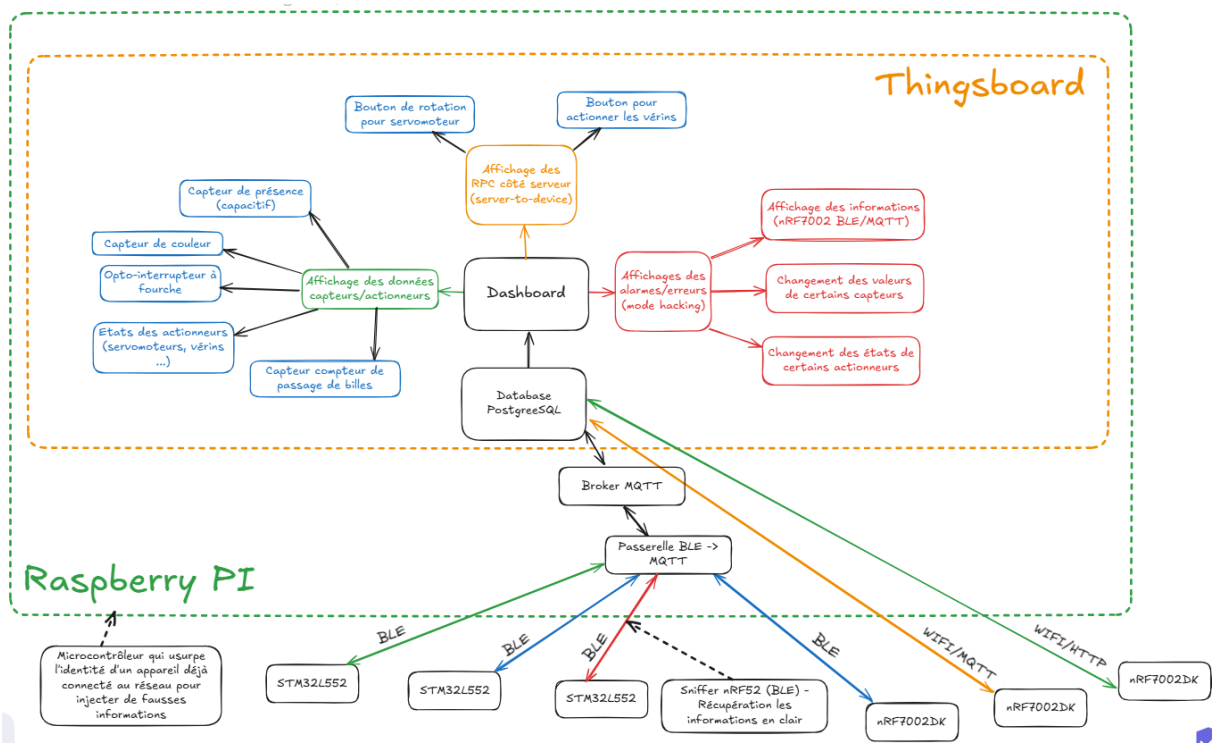


FIGURE 1 – Architecture détaillée du système

Incluant les capteurs, actionneurs, microcontrôleurs et chemins de communication. On y distingue notamment :

- les différents types de capteurs utilisés : capteur de présence, de couleur, opto-interrupteur à fourche, compteur de billes...
- les commandes utilisateurs (via interface ThingsBoard),
- la collecte et l'envoi des données vers le tableau de bord,
- l'activation à distance des actionneurs via les commandes RPC,
- et les voies d'attaque identifiées dans le cadre des tests de cybersécurité.

Cette architecture permet de visualiser en temps réel l'état de l'ensemble du système, d'agir à distance sur celui-ci via ThingsBoard, et d'interagir avec les équipements physiques (ex. : activation de vérins ou servomoteurs via un bouton de dashboard). Le système a été conçu pour s'initialiser automatiquement au démarrage, sans configuration manuelle, ce qui permet une portabilité complète et une mise en œuvre rapide dans un environnement de démonstration.

Au-delà de la supervision et du contrôle, le projet intègre également une dimension expérimentale en cybersécurité.

L'objectif est de démontrer les risques liés à l'absence de sécurisation des échanges dans un environnement IoT typique :

- Un module sniffer BLE (nRF52) est utilisé pour intercepter les données BLE envoyées en clair entre un STM32 et le Raspberry Pi. Cela permet d'illustrer qu'un attaquant passif peut lire les données sensibles (température, humidité, présence...) sans aucune authentification ni chiffrement.
- Un second module est utilisé pour simuler une attaque active de type usurpation d'identité ou MITM (Man-in-the-Middle). Ce composant se fait passer pour un appareil déjà connecté au réseau, en usurpant son adresse MAC et son comportement, dans le but d'injecter de fausses données ou de perturber le système (par ex. déclencher de faux événements, provoquer des erreurs ou des dénis de service).

Ce type de scénario permet d'évaluer les failles de sécurité dans une infrastructure IoT typique et de justifier l'intégration de mécanismes de protection comme TrustZone ou le chiffrement des échanges (décrits dans les sections suivantes).

2.3.2 Liste I/O

Afin d'assurer une organisation claire et structurée de la connectique et de la gestion des capteurs/actionneurs dans notre maquette, nous avons réalisé une liste des entrées/-sorties (I/O) pour chaque microcontrôleur utilisé dans le système.

Chaque microcontrôleur (STM32 ou nRF7002) est associé à une zone bien définie du circuit, comme présenté précédemment. À partir de cette répartition, nous avons identifié les capteurs (entrées) et actionneurs (sorties) reliés à chacun d'eux, en détaillant pour chaque périphérique :

— Le type d'interface
— La tension d'alimentation requise
— L'usage spécifique dans le circuit (ascenseur, détection de bille, actionneur de barrière, etc.)
— Le brochage (pin associé sur le microcontrôleur)
— Les noms des variables dans le code pour la traçabilité

Les tableaux présentés ci-dessous synthétisent ces informations pour chaque objet (zone de la maquette).

	OBJET 1	OBJET 2	OBJET 3	OBJET 4	OBJET 5	OBJET 6
µC	nRF7002DK	nRF7002DK	STM32	STM32	nRF7002DK	STM32
Protocol	WiFi/BLE	WiFi/BLE	BLE (avec shield)	BLE (avec shield)	BLE	BLE (avec shield)
Trustzone Intégré	Oui	Oui	Oui	Oui	Oui	Oui
Alimentation	5V	5V	5V	5V	5V	5V

FIGURE 2 – Configuration matérielle des microcontrôleurs

INPUT

Zone dans le circuit	Capteurs/Input	Référence	Type/Interface	Alimentation	Usage spécifique dans le circuit	Fils/Cablage	Pin(s)	Variables
OBJET 1	Fin de course	KW11	GPIO	3V3 ou 5V	présence lanceur	Alim+GND+TOR	P0.03	O1_FC_1
	Fin de course	KW11	GPIO	3V3 ou 5V	présence lanceur	Alim+GND+TOR	P0.04	O1_FC_2
	Fin de course	KW11	GPIO	3V3 ou 5V	présence lanceur	Alim+GND+TOR	P0.05	O1_FC_3
	Bouton Poussoir	ref non fourni	GPIO/ interruption	12V	activer le lancement	Alim+GND+TOR	P0,06	O1_BP_1
OBJET 2	Fin de course	KW11	GPIO	3V3 ou 5V	Ascenseur en bas	Alim+GND+TOR	P0,09	O2_FC_1
	capteur IR	*TCRT5000*	ADC	*5V*	*présence dans le sas*	Alim+GND+Signal	AIN6 / P0.30	O2_IR_1
OBJET 3	*capteur IR*	*TCRT5000*	ADC	*5V*	*présence dans le sas*	Alim+GND+Signal	PA3	O3_IR_1
	capteur IR	*TCRT5000*	ADC	*5V*	*présence dans le sas*	Alim+GND+Signal	PA2	O3_IR_2
	capteur IR	*TCRT5000*	ADC	*5V*	*présence dans le sas*	Alim+GND+Signal	PC3	O3_IR_3
OBJET 4								
OBJET 5	encodeur rotatif	KY-040	GPIO	5V	Ascenseur en bas	Alim+GND+DT+CLK	P0,08	O5_ER_1
	capteur IR	TCRT5000	ADC	5V	présence dans le sas	Alim+GND+Signal	AIN6 / P0.30	O5_IR_1
	Fin de course	KW11	GPIO	3V3 ou 5V	présence lanceur	Alim+GND+TOR		O5_FC_3
OBJET 6	*Capteur de couleur RGB*	*tcs34725*	I2C	*3V3 à 5V*	*détection de la couleur de la bille*	Alim+GND+SCL+SDA+INT	PB8 / PB9	O6_Color_1

FIGURE 3 – Input

OUTPUT ACTIONNEURS

Zone dans le circuit	Actionneurs	Référence	Type/Interface	Alimentation	Usage spécifique dans le circu	Fils/Cablage	Pin(s)	Variables
OBJET 1	Solénoide	JF-0530B	GPIO	12V ou 24V	Lanceur	Alim+GND	P0.07	O1_SOL_1
	Solénoide	JF-0530B	GPIO	12V ou 24V	Lanceur	Alim+GND	P0.08	O1_SOL_2
	Solénoide	JF-0530B	GPIO	12V ou 24V	Lanceur	Alim+GND	P0.09	O1_SOL_3
	Bande led	WS2815	GPIO	12V	effet de propulsion (lumière)	Alim+GND+DIN+Backup DIN	P0.10	O1_B_LED_1
	Bande led	WS2816	GPIO	12V	effet de propulsion (lumière)	Alim+GND+DIN+Backup DIN	P1.01	O1_B_LED_2
	Haut parleur	Aiyima 3ohm 4w + MAX98357A	I2S	5V	Effet sonore au lancement	SCK+LRCK+SDIN+SDOUT	P0.29 / P0.28 / P0.03 / P0.02	O1_HP_1
OBJET 2	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	P0.05	O2_Servo_1
	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	P0.06	O2_Servo_2
	Actionneur linéaire/ Vérin électrique	Pas de ref précise	GPIO	12V	Ascenseur	Alim+GND+TOR	P0.04	O2_Verin_1
	Bande led	WS2816	GPIO	12V	effet de propulsion (lumière)	Alim+GND+DIN+Backup DIN	P0.07 / P0.8	O2_B_LED_1
	Haut parleur	Aiyima 3ohm 4w + MAX98357A	I2S	5V	Effet sonore au lancement	SCK+LRCK+SDIN+SDOUT	P0.29 / P0.28 / P0.03 / P0.02	O2_HP_1
OBJET 3	Servomoteur	MG90S	PWM	5V	Aiguillage	Alim+GND+PWM	PC8	O3_Servo_1
	Servomoteur	MG90S	PWM	5V	Aiguillage	Alim+GND+PWM	PC6	O3_Servo_2
	Led RGB		GPIO	12V	Etat de l' aiguillage (lumière)	Alim+GND+R+G	PC0/PC1	O3_LED_1
	Led RGB		GPIO	12V	Etat de l' aiguillage (lumière)	Alim+GND+R+G	PC2/PC4	O3_LED_2
OBJET 4	Bumper			24V				O3_BMP_1
	Bumper			24V				O3_BMP_2
	Bumper			24V				O3_BMP_3
	Bumper			24V				O3_BMP_4
OBJET 5	Haut parleur	Aiyima 3ohm 4w + MAX98357A	I2S	5V	Effet sonore au lancement	SCK+LRCK+SDIN+SDOUT	P0.29 / P0.28 / P0.03 / P0.02	O5_HP_1
	Bande led	WS2816	GPIO	12V	effet de propulsion (lumière)	Alim+GND+DIN+Backup DIN	P0.07	O5_B_LED_1
	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	P0.05	O5_Servo_1
	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	P0.06	O5_Servo_2
	Actionneur linéaire/ Vérin électrique	Pas de ref précise	GPIO	12V	Ascenseur	Alim+GND + TOR	P0.04	O5_Verin_1
	afficheur 7 segment 2 chiffres	x	x	x	x	x	x	x
OBJET 6	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	PC8	O5_Servo_1
	Servomoteur	MG90S	PWM	5V	barrière	Alim+GND+PWM	PC6	O5_Servo_2

FIGURE 4 – Output

La réalisation de cette table nous a permis de planifier précisément l'achat de capteurs compatibles en termes d'alimentation et d'interface.

2.3.3 Fonctionnement détaillé d'un microcontrôleur dans la chaîne de traitement

Afin d'illustrer concrètement le rôle d'un microcontrôleur dans la chaîne de traitement de notre maquette, nous avons modélisé en détail le fonctionnement d'un des microcontrôleurs responsables de la gestion d'une zone du circuit.

La figure suivante présente les connexions physiques du microcontrôleur avec ses capteurs et actionneurs. On y retrouve des entrées issues de capteurs de fin de course, d'un bouton poussoir et de timers internes, ainsi que des sorties vers des bandes LED, des solénoïdes, un haut-parleur via I2S, et une interface de communication BLE/USB. Ces éléments sont pilotés par des ports GPIO, des interfaces SPI/I2S, ou encore des interruptions matérielles.

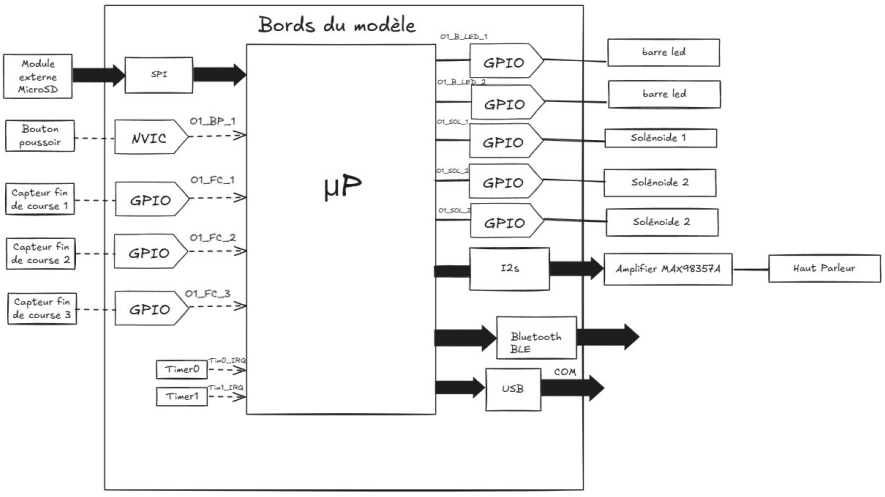


FIGURE 5 – Bords du modèle

Pour donner une vision plus dynamique du traitement interne, deux autres représentations ont été construites : le flot d'événements et le flot de données.

Le flot d'événements illustre la manière dont le microcontrôleur réagit aux interruptions déclenchées par les capteurs (par exemple, lorsqu'une bille passe devant un capteur IR ou actionne un fin de course). Ces événements déclenchent l'exécution de tâches spécifiques, telles que l'activation d'un solénoïde ou le démarrage d'un effet lumineux.

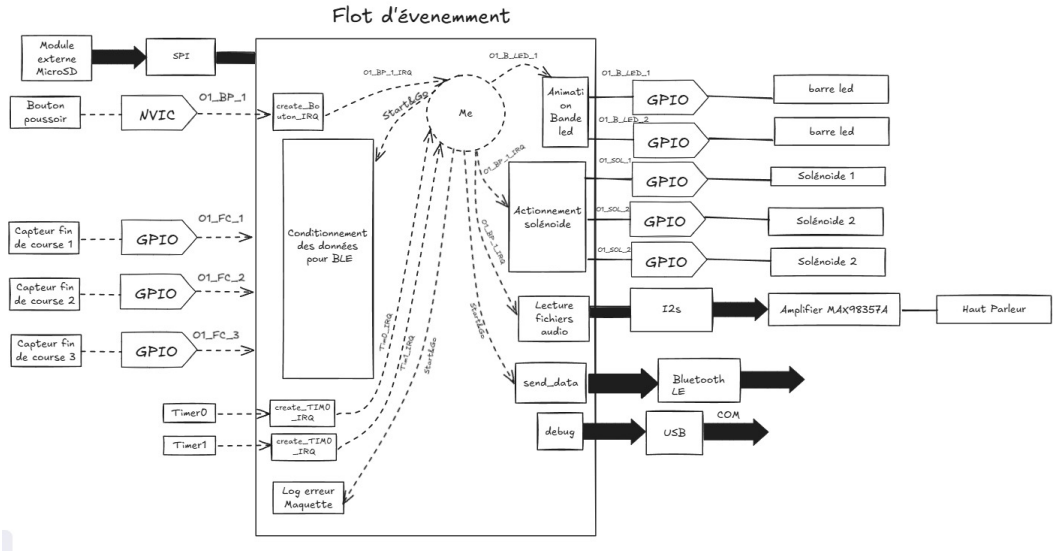


FIGURE 6 – Flot d'événements

Le flot de données met en avant les informations périodiques envoyées par le microcontrôleur, comme les logs d'erreurs, les états de capteurs/actionneurs ou encore les trames de données BLE à destination du serveur ThingsBoard

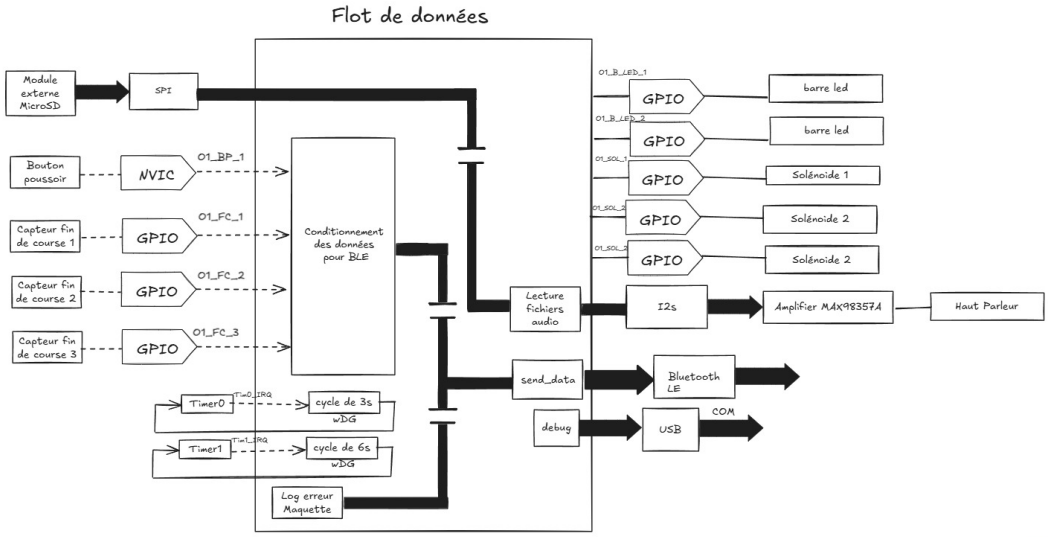


FIGURE 7 – Flot de données

2.4 Choix des composants matériels et logiciels

Concernant le choix des composants matériels et logiciels, nous avons commencé par sélectionner le tableau de bord sur lequel reposerait l'interface de supervision. Ce choix devait répondre à plusieurs critères essentiels : une interface simple d'utilisation, esthétique et épurée, une compatibilité avec les protocoles MQTT et HTTP, ainsi que la possibilité de le déployer en local ou sur le cloud, selon l'évolution du projet.

Aspect techniques				
Application	**Type de passerelle recommandé**	**Protocole(s) supporté(s)**	**Déploiement**	**Avis esthétique**
ThingsBoard	Ethernet / Wi-Fi / LoRaWAN	MQTT, HTTP, CoAP	Local / Cloud	Claire visuellement
Grafana	Ethernet / Wi-Fi	MQTT, HTTP	Local / Cloud	Moins claire visuellement
Kaa IoT Platform	Ethernet / Wi-Fi / 4G LTE	MQTT, HTTP	Cloud	Claire visuellement
Mainflux	Ethernet / Wi-Fi / LoRaWAN	MQTT, HTTP, CoAP	Local / Cloud	Graphique vieillissant
Node-RED	Ethernet / Wi-Fi / Bluetooth / Zigbee	MQTT, HTTP, WebSocket	Local	Scolaire mais simple et rapide d'utilisation
Machinechat JEDI	Ethernet / Wi-Fi	MQTT, HTTP	Local	Claire visuellement
Ubidots	Ethernet / Wi-Fi / 4G LTE	MQTT, HTTP	Cloud	Clair mais uniquement en cloud
CODESYS	Ethernet / Wi-Fi / Modbus	Modbus, OPC UA, MQTT	Local	Ancien et axé supervision industrielle

FIGURE 8 – Comparaison des solutions de tableau de bord

Après avoir comparé différentes solutions, nous avons choisi ThingsBoard, car il répondait à l'ensemble des critères que nous avions définis : une interface esthétique et épurée, la prise en charge des protocoles MQTT, HTTP et CoAP, ainsi que la possibilité de le déployer aussi bien en local que sur le cloud.

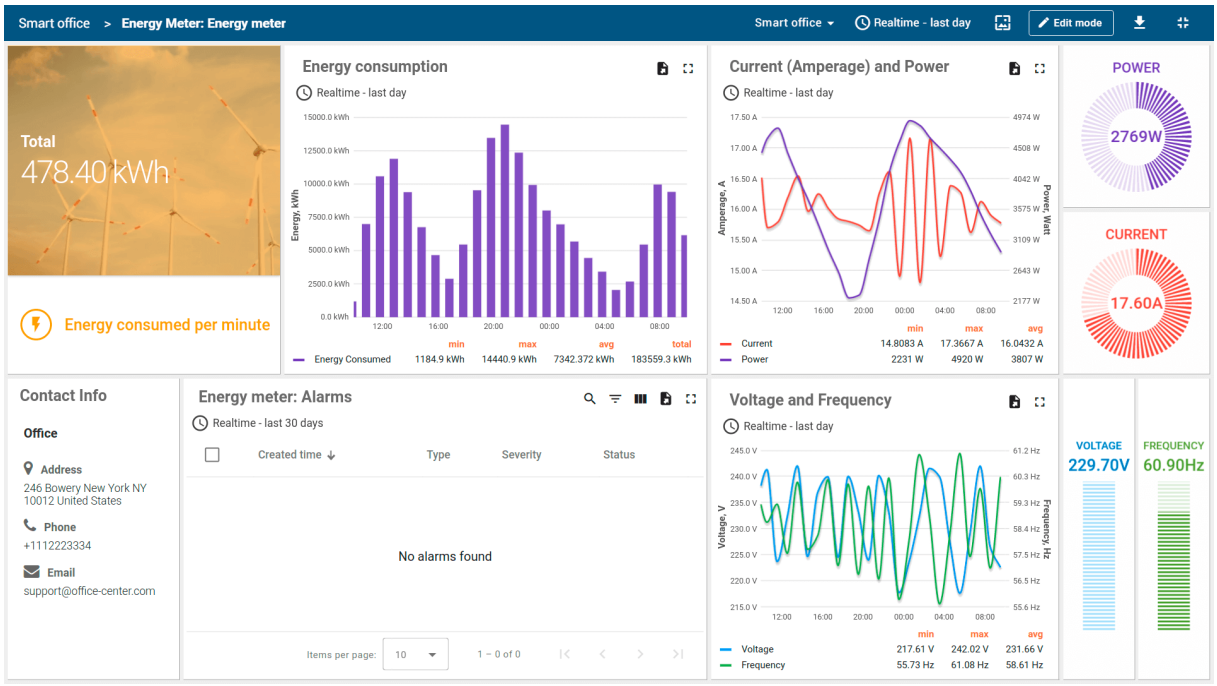
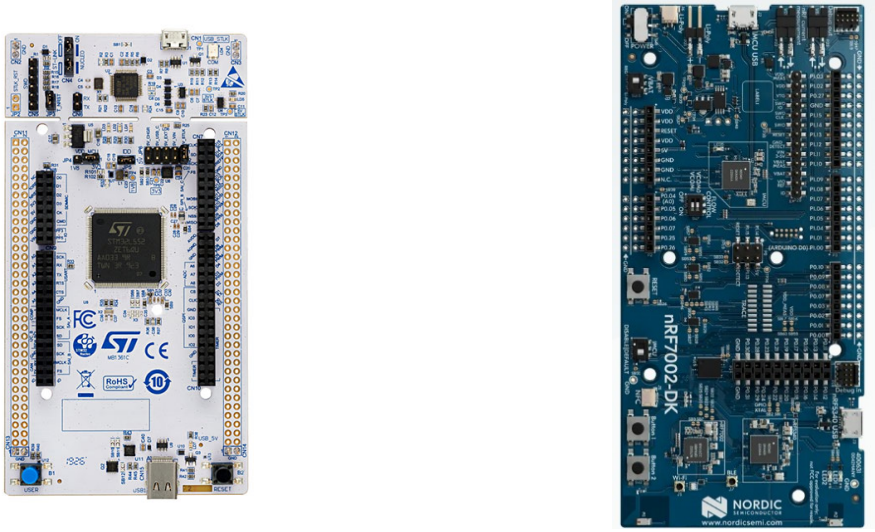


FIGURE 9 – Exemple d'un tableau de bord Thingsboard

Concernant le choix des microcontrôleurs, celui-ci nous a été imposé. Nous avons utilisé des microcontrôleurs STM32L552 ainsi que des microcontrôleurs Nordic Semiconductor nRF5340. Ce dernier est couplé à un module nRF7002, un coprocesseur Wi-Fi qui fonctionne avec le nRF5340 et qui assure la connectivité Wi-Fi. Ce choix permet d'utiliser deux différentes technologies. De plus, un module BLE a été utilisé pour le STM32L552.



(a) STM32L552 (b) nRF7002

FIGURE 10 – Cartes microcontrôleurs utilisées dans le projet : STM32L552 et nRF7002

Nous devions également choisir comment héberger notre serveur. Bien qu'un ordinateur classique nous ait été proposé, nous avons rapidement écarté cette option en raison des contraintes de place et de transportabilité liées à la maquette. Nous avons donc décidé d'opter pour un Raspberry Pi, solution compacte et plus adaptée.

Dans un premier temps, la version 3 du Raspberry Pi nous a été suggérée. Cependant, après l'avoir testée, nous avons constaté que sa mémoire vive, limitée à 1 Go de RAM, était insuffisante pour faire fonctionner de manière fluide les services nécessaires (ThingsBoard, broker MQTT, base de données...). Nous avons alors choisi de passer à un Raspberry Pi 5, équipé de 4 Go de RAM, ce qui s'est révélé bien plus adapté aux besoins en ressources du système.

(a) Raspberry PI 5 (b) Module BLE Nucleo-BNRG2A1

FIGURE 11 – Raspberry PI 5 et module Bluetooth Low Energy pour STM32L552

3 Gestion de projet

3.1 Diagramme de Gantt et méthodes de travail

La planification du projet a constitué un pilier fondamental de notre démarche. Afin d'organiser efficacement les différentes étapes, un diagramme de Gantt a été élaboré dès le début du stage

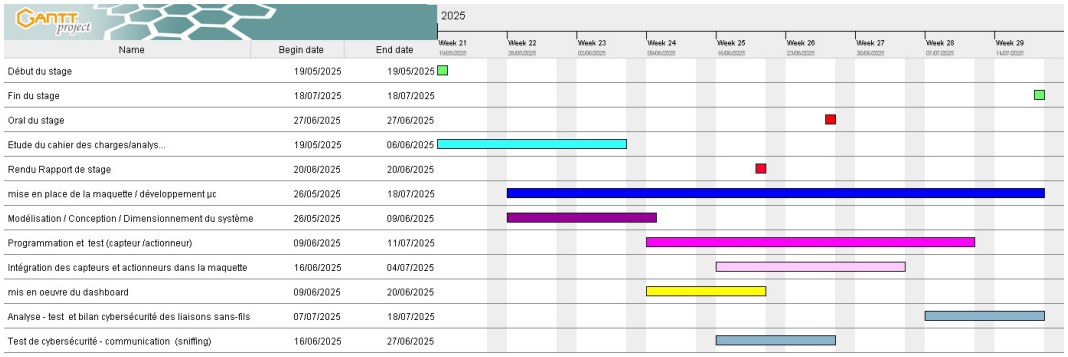


FIGURE 12 – Diagramme de Gantt

Celui-ci permettait de visualiser l'enchaînement logique des tâches, leur durée estimée ainsi que les éventuelles périodes de chevauchement. Cette visualisation a facilité une gestion fluide des priorités, tout en offrant la souplesse nécessaire pour faire face aux imprévus techniques ou matériels.

Tout au long du projet, nous avons adopté une approche de travail structurée, collaborative et évolutive, dans le but de garantir une progression continue, malgré les aléas rencontrés.

Nous organisions au moins une réunion hebdomadaire. Ces réunions avaient pour objectifs de :

- faire le point sur l'avancement des tâches de chacun,
- partager les idées et proposer de nouvelles pistes d'amélioration,
- valider la faisabilité technique de certaines décisions,
- et réajuster, si besoin, la répartition des responsabilités.

Un des principes essentiels de notre organisation consistait à éviter les blocages prolongés : lorsqu'un membre rencontrait une difficulté technique, nous choisissions collectivement de mettre la tâche temporairement de côté, afin de concentrer nos efforts sur d'autres objectifs réalisables. Cette stratégie nous a permis de maintenir un rythme de développement régulier, tout en revenant plus tard, avec recul ou nouvelles idées, sur les points bloquants.

Enfin, nous avons fait preuve d'une réelle solidarité et flexibilité : lorsqu'un membre était surchargé ou rencontrait un blocage, un autre pouvait prendre le relais sur sa tâche. Cette dynamique de coopération a grandement contribué à la réussite globale du projet, en favorisant une entraide constante et une meilleure cohésion au sein du groupe.

4 Développement du projet

4.1 Mise en place du point d'accès sur Raspberry

Dans l'architecture de la maquette, le Raspberry Pi devait assurer la centralisation des données échangées entre les microcontrôleurs et le tableau de bord ThingsBoard. Étant donné que le système devait fonctionner en réseau local autonome, sans connexion à Internet ni à un réseau externe, il était nécessaire de transformer le Raspberry Pi en point d'accès Wi-Fi.

Avant toute chose, il a fallu installer un système d'exploitation sur le Raspberry Pi. Pour cela, nous nous sommes appuyés sur le tutoriel fourni en début d'année par Hervé Boeglen, dans le cadre du cours de gestion de projet informatique.

Nous avons utilisé le logiciel Raspberry Pi Imager, qui permet d'installer facilement un système d'exploitation sur une carte SD.

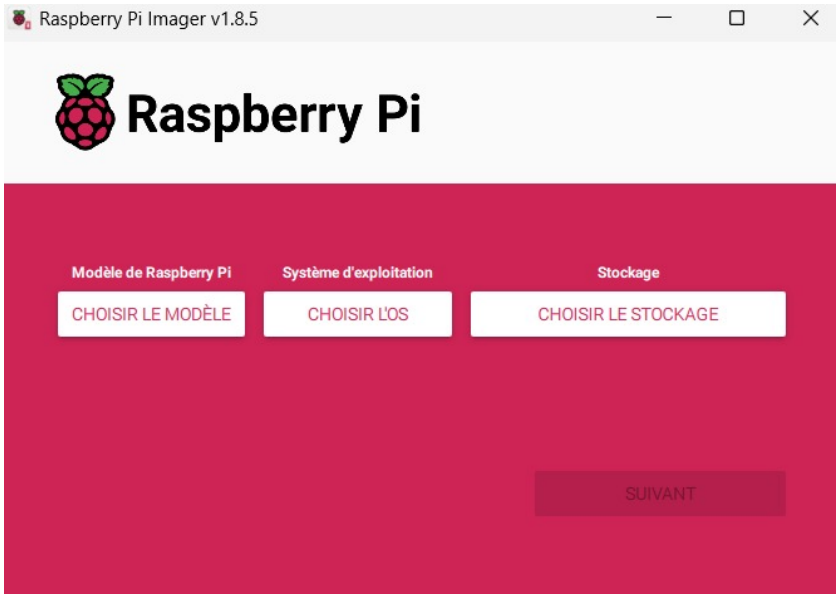


FIGURE 13 – Logiciel Raspberry Pi Imager

Après avoir sélectionné notre modèle de Raspberry Pi (Raspberry Pi 5), nous avons choisi le système Raspberry Pi OS (64 bits), puis indiqué la carte SD comme support d'installation.

Une fois l'OS correctement installé sur la carte SD, nous avons extrait la carte microSD et l'avons insérée dans le Raspberry Pi. Nous avons ensuite connecté un écran, un clavier et une souris au Raspberry Pi, avant de procéder à sa mise sous tension.

Ensuite nous avons procédé à la mise en place du point d'accès Wi-Fi, afin de créer un réseau local isolé. Dans un premier temps, nous avons tenté de configurer manuellement le point d'accès à l'aide des outils classiques (hostapd, dnsmasq, etc.), mais cette méthode s'est révélée complexe et instable. Plusieurs problèmes sont apparus, notamment des conflits de services, des erreurs DHCP et un démarrage aléatoire du réseau.

Pour simplifier la configuration et garantir une meilleure stabilité, nous avons choisi de passer par une méthode plus moderne en utilisant NetworkManager. Grâce à son interface graphique nous avons pu créer plus facilement un point d'accès Wi-Fi.

FIGURE 14 – Interface graphique pour créer un point d'accès Wi-Fi

Nous avons créé un point d'accès avec le SSID : RaspberryPi0. Ce réseau a été sécurisé à l'aide du protocole WPA2, ce qui permet de chiffrer les échanges entre les appareils connectés et de protéger l'accès par mot de passe. Lors de la configuration, le système nous a automatiquement proposé de définir un mot de passe pour renforcer la sécurité du réseau.

Le protocole de sécurité WPA2 n'a pas été choisi par hasard pour protéger le point d'accès Wi-Fi. Nous avons dans un premier temps testé WPA3, plus récent et plus sécurisé, mais nous avons rencontré des problèmes de compatibilité. En effet, lors de nos tests, les microcontrôleurs nRF7002 n'arrivaient pas à se connecter au point d'accès lorsque celui-ci était configuré en WPA3. En revanche, dès que nous sommes repassés à WPA2, la connexion s'est établie sans difficulté. En réalité, sur la Figure 7, l'option WPA/WPA2 Personal n'était pas directement disponible. À l'origine, seule l'option WPA/WPA2/WPA3 Personal était proposée. Afin de forcer l'utilisation exclusive de WPA2, nous avons dû modifier manuellement les fichiers de configuration du point d'accès.

Pour cela, nous nous sommes rendus dans le répertoire suivant :

```
/etc/NetworkManager/system-connections/
```

Puis nous avons ouvert le fichier correspondant à notre point d'accès, ici nommé RaspberryPi0.nmconnection. À l'intérieur de ce fichier, dans la section [wifi-security], nous avons ajouté la ligne suivante :

```
proto=rsn
```

Cette directive permet de restreindre les protocoles de sécurité utilisés à WPA2 uniquement (RSN : Robust Security Network), en excluant WPA et WPA3.

FIGURE 15 – Fichier RaspberryPi0.nmconnection

La différence entre WPA2 et WPA3 réside principalement dans le mécanisme d'authentification et le niveau de sécurité. WPA2 (Wi-Fi Protected Access 2) repose sur le chiffrement AES et utilise un échange de clé basé sur le protocole PSK (Pre-Shared Key). À l'inverse, WPA3, plus récent, renforce la sécurité en remplaçant le PSK par le protocole SAE (Simultaneous Authentication of Equals), qui offre une meilleure résistance aux attaques par dictionnaire et aux tentatives d'accès non autorisé.

Ce choix a donc été motivé par une contrainte de compatibilité matérielle, tout en conservant un niveau de sécurité suffisant.

Pour vérfier si le point d'accès était actif il a fallu exécuter la commande suivante :

```
sudo systemctl status NetworkManager
```

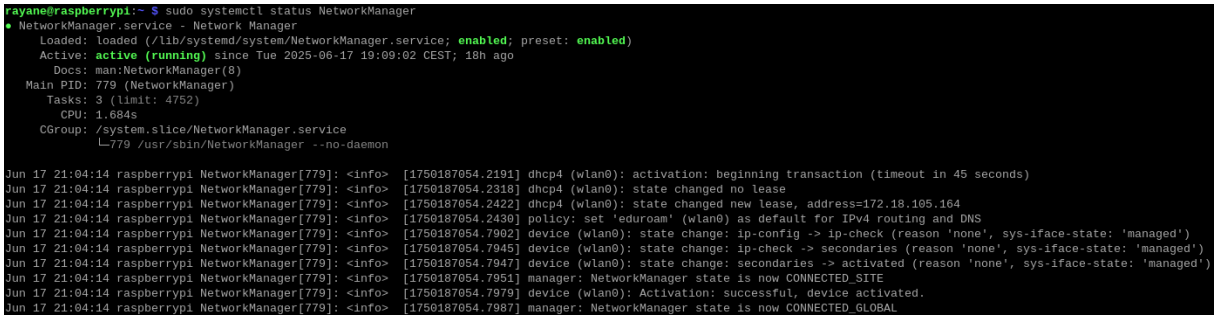


FIGURE 16 – État du service NetworkManager

Sur la capture d'écran, on peut voir que le statut est affiché comme active (running), ce qui signifie que le point d'accès est bien opérationnel et que le service NetworkManager est en marche.

4.2 Création du tableau de bord Thingsboard

Dans le cadre du projet, nous avons utilisé la plateforme ThingsBoard pour centraliser, afficher et interagir avec les données transmises par les microcontrôleurs. ThingsBoard permet de créer un tableau de bord graphique, d'afficher des valeurs en temps réel, de piloter des actionneurs à distance et de simuler différents scénarios IoT.

L'installation s'est réalisée en ligne de commande. Nous avons tout d'abord installé Java, car ThingsBoard est une application développée en Java et repose sur une machine virtuelle Java (JVM) pour fonctionner. Le serveur backend de ThingsBoard utilise ce runtime pour exécuter ses services : gestion des appareils, interactions avec la base de données, communications MQTT/HTTP, traitement des données, etc.

L'installation de Java s'est effectuée avec la commande suivante :

```
sudo apt update && sudo apt install openjdk-17-jdk
```

Après l'installation de Java, nous avons procédé à l'installation du service ThingsBoard lui-même. Le paquet d'installation a été téléchargé directement depuis le dépôt officiel GitHub à l'aide de la commande suivante :

```
wget https://github.com/thingsboard
/thingsboard/releases/download/v4.0.1/thingsboard-4.0.1.deb
```

Une fois le fichier .deb obtenu, nous avons installé ThingsBoard en tant que service sur le système à l'aide de la commande :

```
sudo dpkg -i thingsboard-4.0.1.deb
```

Cela a permis d'enregistrer ThingsBoard comme service système, ce qui facilite son démarrage automatique au boot du Raspberry Pi.

Après l'installation du service, nous avons procédé à la configuration de la base de données PostgreSQL, indispensable au fonctionnement de ThingsBoard. Cette base est utilisée pour stocker toutes les informations liées aux appareils, aux utilisateurs, aux tableaux de bord, ainsi qu'aux données télémétriques envoyées par les microcontrôleurs.

Ensuite, nous avons ajouté la clé de signature du dépôt officiel PostgreSQL, puis référencé ce dépôt dans la configuration de notre système de paquets, à l'aide des commandes suivantes :

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |
    sudo apt-key add -

echo "deb https://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)
   -pgdg main" | sudo tee/etc/apt/sources.list.d/pgdg.list
```

Ces étapes ont permis au système de reconnaître les paquets PostgreSQL officiels. Nous avons ensuite mis à jour la liste des paquets disponibles et installé PostgreSQL avec les commandes suivantes :

```
sudo apt update

sudo apt -y install postgresql
```

Puis nous avons lancé le service postgresql avec la commande suivante :

```
sudo service postgresql start
```

Une fois PostgreSQL installé, il est recommandé de sécuriser le système en définissant un mot de passe. Cela se fait à l'aide de la commande suivante :

```
sudo -u postgres psql -c "\password"
```

Il faut ensuite entrer et confirmer ce mot de passe.

Après avoir défini le mot de passe, nous nous sommes connectés manuellement à la base PostgreSQL avec cette commande :

```
psql -U postgres -d postgres -h 10.42.0.1 -W
```

Cette commande signifie que nous nous connectons à la base nommée postgres, en tant qu'utilisateur postgres, via l'hôte local (10.42.0.1), et que le système demandera le mot de passe (-W).

Une fois connectés à la console PostgreSQL, nous avons créé la base de données utilisée par ThingsBoard à l'aide de la commande SQL suivante :

```
CREATE DATABASE thingsboard;
```

Après avoir créé la base de données, il a été nécessaire de configurer ThingsBoard afin qu'il puisse s'y connecter correctement. Pour cela, nous avons modifié le fichier de configuration principal de l'application, accessible via la commande suivante :

```
sudo nano /etc/thingsboard/conf/thingsboard.conf
```

Dans ce fichier, nous avons ajouté les lignes suivantes pour indiquer à ThingsBoard d'utiliser PostgreSQL comme moteur de base de données :

```
export DATABASE_TS_TYPE=sql exportSPRING_DATASOURCE_URL=jdbc:
   postgresql://localhost:5432/thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=MOT_DE_PASSE
export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS

export JAVA_OPTS="$JAVA_OPTS -Xms2G -Xmx2G"
```

Cette configuration permet à ThingsBoard de se connecter automatiquement à la base de données PostgreSQL créée précédemment. Elle définit notamment :

le type de stockage utilisé (sql), l'adresse de la base (localhost :5432/thingsboard), les identifiants de connexion (postgres + mot de passe défini manuellement) et la politique de partitionnement des données temporelles (mensuelle).

La dernière ligne configure également l'allocation de mémoire pour la machine virtuelle Java exécutant ThingsBoard. Ici, le service démarre avec un minimum de 2 Go de mémoire et peut monter jusqu'à 2 Go, ce qui est adapté à notre usage sur Raspberry Pi.

Une fois cette étape terminée, nous avons démarré ThingsBoard en tant que service à l'aide de la commande :

```
sudo service thingsboard start
```

Et nous avons vérifié le status du service à l'aide de la commande :

```
sudo service thingsboard status
```

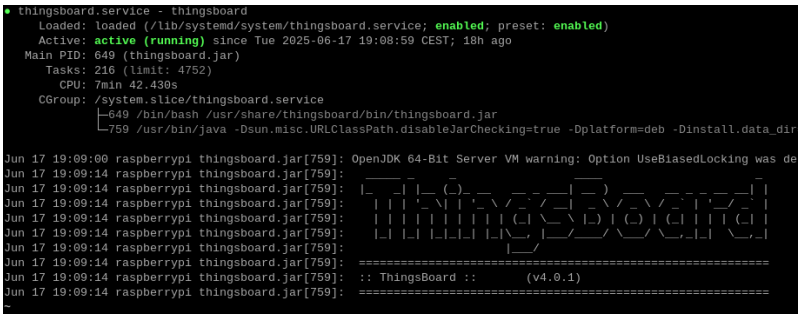


FIGURE 17 – État du service Thingsboard

Une fois le service ThingsBoard actif, nous avons ouvert un navigateur afin de nous connecter à l'adresse : **localhost :8080**. Cela nous a redirigé vers la page suivante :

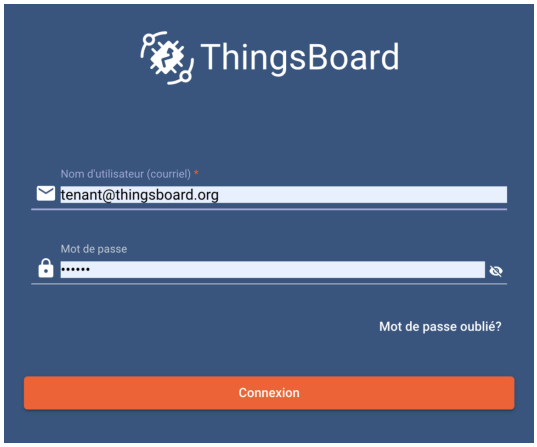


FIGURE 18 – Page de connexion Thingsboard

Nous devions nous connecter en tant que Tenant, ce qui nous permet d'accéder à l'espace utilisateur principal dans ThingsBoard, celui destiné à gérer concrètement les objets connectés, les dashboards, les widgets, etc. Enfin, nous avons créé un tableau de bord prototype afin de tester les différentes fonctionnalités offertes par ThingsBoard.

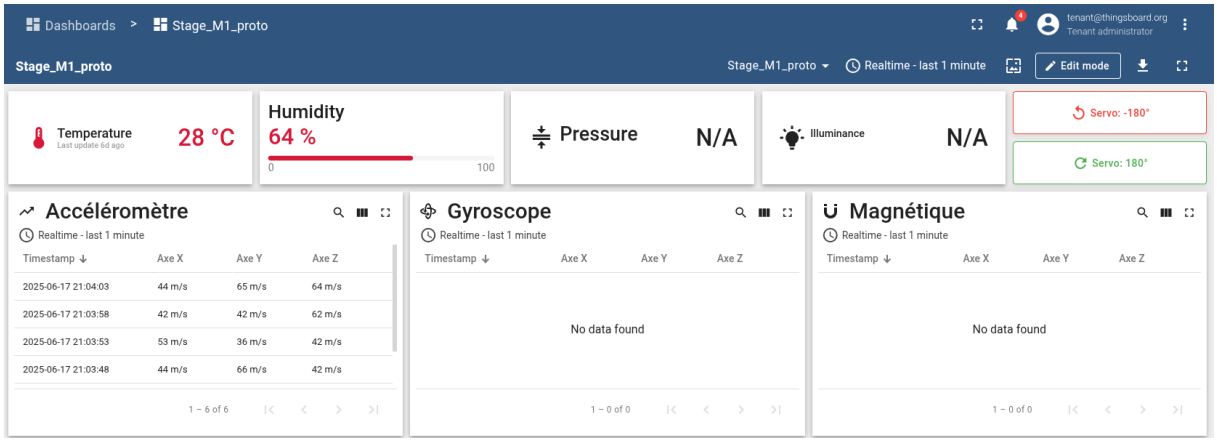


FIGURE 19 – Dashboard Prototype Thingsboard

4.3 Transmission des données vers ThingsBoard via Wi-Fi/HTTP

Le protocole HTTP (HyperText Transfer Protocol) est couramment utilisé pour la communication entre les navigateurs web et les serveurs. Dans le contexte de l'Internet des Objets (IoT), il est parfois adopté pour sa simplicité d'implémentation, car il permet une intégration rapide avec des plateformes comme ThingsBoard via son API REST, et bénéficie d'un support natif dans de nombreux systèmes embarqués et middlewares, tels que Zephyr.

Cependant, HTTP présente plusieurs limites importantes en matière de cybersécurité comme l'absence de chiffrement par défaut, transmission des données (y compris les tokens d'authentification) en clair sur le réseau, vulnérabilité aux attaques de type Man-in-the-Middle (MITM).

Dans un contexte réel, HTTP doit impérativement être remplacé par HTTPS, sa version sécurisée, qui chiffre les échanges et garantit l'authenticité du serveur.

Nous avons utilisé un microcontrôleur nRF7002 avec le système d'exploitation Zephyr RTOS pour établir une communication entre l'objet connecté et la plateforme ThingsBoard, hébergée sur un Raspberry Pi.

Le programme réalise les étapes suivantes :

Le nRF7002 scanne les réseaux disponibles, puis tente de se connecter au point d'accès local "RaspberryPi0". Une fois connecté, il obtient une adresse IP via DHCP (ici 10.42.0.96), ce qui permet d'établir des communications réseau avec le serveur ThingsBoard.

```
[00:00:30.551,330] <inf> wifi_monitor: Connexion au réseau Wi-Fi RaspberryPi0...
[00:00:30.559,204] <inf> wifi_monitor: Scan des r◆◆seaux Wi-Fi...
[00:00:35.180,572] <inf> wifi_monitor: Réseau détecté : SSID=RaspberryPi0, RSSI=-47 dBm, canal=6
[00:00:35.190,093] <inf> wifi_monitor: R◆◆seau détecté : SSID=eduroam, RSSI=-64 dBm, canal=6
[00:00:35.199,188] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP, RSSI=-64 dBm, canal=6
[00:00:35.208,282] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP-BIS, RSSI=-73 dBm, canal=100
[00:00:35.217,926] <inf> wifi_monitor: Réseau détecté : SSID=eduroam, RSSI=-74 dBm, canal=100
[00:00:35.227,203] <inf> wifi_monitor: Réseau détecté : SSID=eduroam, RSSI=-79 dBm, canal=1
[00:00:35.236,297] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP, RSSI=-81 dBm, canal=11
[00:00:35.245,483] <inf> wifi_monitor: Réseau détecté : SSID=eduroam, RSSI=-83 dBm, canal=11
[00:00:35.254,699] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP, RSSI=-83 dBm, canal=1
[00:00:35.263,824] <inf> wifi_monitor: Réseau détect◆◆ : SSID=eduroam, RSSI=-86 dBm, canal=1
[00:00:35.272,918] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP, RSSI=-87 dBm, canal=1
[00:00:35.282,012] <inf> wifi_monitor: R◆◆seau détecté : SSID=ED-SP2MI, RSSI=-88 dBm, canal=6
[00:00:35.291,198] <inf> wifi_monitor: Réseau détecté : SSID=WIFI-UP, RSSI=-93 dBm, canal=1
[00:00:35.300,262] <inf> wifi_monitor: Scan Wi-Fi terminé.
[00:00:38.463,928] <inf> wifi_monitor: Résultat de la connexion Wi-Fi : status = 0
[00:00:38.471,954] <inf> wifi_monitor: Connecté au Wi-Fi !
[00:00:38.496,063] <inf> net_dhcpv4: Received: 10.42.0.96
[00:00:38.501,983] <inf> net_config: IPv4 address: 10.42.0.96
[00:00:38.508,117] <inf> net_config: Lease time: 3600 seconds
[00:00:38.514,312] <inf> net_config: Subnet: 255.255.255.0
[00:00:38.520,202] <inf> net_config: Router: 10.42.0.1
```

FIGURE 20 – Logs de la connexion au point d'accès Raspberry

Afin de simuler un capteur de température et d'humidité, le microcontrôleur génère aléatoirement deux valeurs. Ces données sont ensuite formatées au format JSON, un format léger et lisible de représentation de données. Il permet de structurer les informations sous forme de paires clé/valeur, faciles à analyser, transmettre et afficher.

```c
// Simule des données de capteur
static void get_sensor_data(char *json_buffer, size_t buffer_size)
{
    int16_t temp = (sys_rand32_get() % 40) - 10;
    int16_t hum = sys_rand32_get() % 100;

    snprintf(json_buffer, buffer_size,
             "{\"temp\":%d,\"hum\":%d}",
             temp, hum);
}
```

FIGURE 21 – Extrait de code pour la simulation de données

Une requête HTTP POST est construite manuellement dans le code (headers + corps) et envoyée via une socket TCP vers le serveur ThingsBoard, à l'adresse :

`http://10.42.0.1:8080/api/v1/vYl0k1gcVLlJE6TTTggs/telemetry`

Le jeton d'accès (token) est unique pour chaque dispositif et permet à ThingsBoard d'identifier l'émetteur.

Une fois la requête envoyée, le programme lit la réponse retournée par ThingsBoard (code HTTP, confirmation) et l'affiche via le module de logs de Zephyr.

```
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options:
[00:00:43.819,732] <inf> wifi_monitor: Données envoy◆◆es avec succès!
[00:00:53.829,071] <inf> wifi_monitor: Connexion à ThingsBoard 10.42.0.1:8080
[00:00:53.883,361] <inf> wifi_monitor: Réponse reçue: HTTP/1.1 200
```

FIGURE 22 – Confirmation de la réception de données

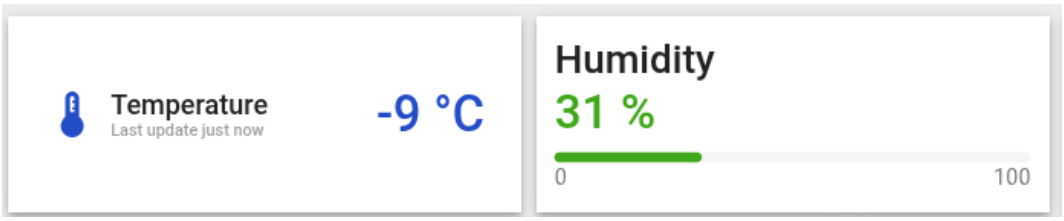


FIGURE 23 – Affichage des données simulées sur Thingsboard

4.4 Transmission des données vers ThingsBoard via Wi-Fi/MQTT et contrôle par RPC

Le protocole MQTT (Message Queuing Telemetry Transport) est largement utilisé dans le domaine de l'IoT en raison de sa légèreté, de son efficacité énergétique et de son modèle de communication basé sur le publish/subscribe. Contrairement à HTTP, où chaque appareil doit faire une requête au serveur, MQTT permet à un broker central (Mosquitto) de distribuer les messages uniquement aux clients abonnés à un topic, réduisant ainsi le trafic réseau inutile.

Chaque microcontrôleur publie ses données sur un topic MQTT spécifique, qui est ensuite intercepté par le broker Mosquitto tournant sur le Raspberry Pi. ThingsBoard est configuré pour s'abonner à ces topics et traiter automatiquement les messages reçus.

L'installation et la configuration de Mosquitto, ainsi que le script Python permettant de créer un broker MQTT, ayant été réalisés en amont, nous avons décidé de développer un programme permettant d'envoyer des données vers ThingsBoard en utilisant le protocole MQTT.

Dans un premier temps, nous avons développé un programme embarqué permettant de simuler l'envoi de données issues d'un gyroscope (les vitesses de rotation selon trois axes) via le protocole MQTT. Ces données sont publiées sur un topic spécifique auquel est abonné le broker Mosquitto, configuré pour les relayer à ThingsBoard à travers la passerelle BLE vers MQTT.

Un thread sensor publish thread() est responsable de simuler des données de gyroscope, en générant toutes les 5 secondes trois valeurs aléatoires représentant les axes Axe X, Axe Y et Axe Z. Ces données sont formatées au format JSON, puis transmises à ThingsBoard via un topic MQTT, défini dans la configuration.

```c
void sensor_publish_thread(void *arg1, void *arg2, void *arg3)
{
    struct mqtt_client *client = (struct mqtt_client *)arg1;
    char payload[128];

    while (1) {
        int Axe_X = 30 + (sys_rand32_get() % 40);
        int Axe_Y = 30 + (sys_rand32_get() % 40);
        int Axe_Z = 30 + (sys_rand32_get() % 40);

        snprintk(payload, sizeof(payload),
                 "{\"Axe_X\": %d, \"Axe_Y\": %d, \"Axe_Z\": %d}",
                 Axe_X, Axe_Y, Axe_Z);

        int err = data_publish(client, MQTT_QOS_1_AT_LEAST_ONCE,
                               (uint8_t *)payload, strlen(payload));
        if (err) {
            printk("MQTT publish error: %d", err);
        } else {
            printk("Published sensor data: %s", payload);
        }

        k_sleep(K_SECONDS(5));
    }
}
```

FIGURE 24 – Extrait de code qui montre la publication des données

Dans un second temps, nous avons enrichi ce programme en y intégrant la gestion des RPC (Remote Procedure Call). Cette fonctionnalité permet à l'utilisateur, directement depuis le tableau de bord ThingsBoard, d'interagir avec le microcontrôleur distant. Concrètement, deux boutons ont été ajoutés à l'interface afin de piloter un servomoteur à distance. Selon l'action envoyée (gauche ou droite), le microcontrôleur exécute la commande et oriente le servomoteur dans la direction souhaitée.

En parallèle, le microcontrôleur est abonné à un topic spécifique utilisé par ThingsBoard pour envoyer des commandes RPC (v1/devices/me/rpc/request/+). Lorsqu'une commande est reçue, elle est automatiquement parsée (au format JSON) grâce à la bibliothèque cJSON. Si la commande correspond à une méthode setServo, le microcontrôleur récupère l'angle passé en paramètre, puis ajuste le servomoteur en conséquence.

L'extrait de code suivant montre la logique de traitement d'une commande RPC envoyée depuis ThingsBoard. Le microcontrôleur utilise la bibliothèque `cJSON` pour parser le message, vérifie que la méthode est bien `setServo`, puis applique la valeur d'angle reçue pour ajuster le servomoteur :

```
// ====== Ajout RPC JSON ThingsBoard ======
cJSON *root = cJSON_Parse((char*)payload_buf);
if (root) {
    cJSON *method = cJSON_GetObjectItem(root, "method");
    if (method && strcmp(method->valuestring, "setServo") == 0) {
        cJSON *params = cJSON_GetObjectItem(root, "params");
        if (params) {
            cJSON *angle = cJSON_GetObjectItem(params, "angle");
            if (angle) {
                printk(">> COMMANDE RPC ThingsBoard : setServo %d\n", angle->valueint);
                set_servo_pulse(angle_to_pulse(angle->valueint));
            }
        }
    }
```

FIGURE 25 – Traitement d'une commande RPC reçue via MQTT

Le servomoteur utilisé est alimenté directement par la carte nRF7002 DK, à partir de sa sortie 5V. Il est relié à la masse (GND) de la carte et reçoit les impulsions de commande via la broche P1.07, configurée en sortie PWM.

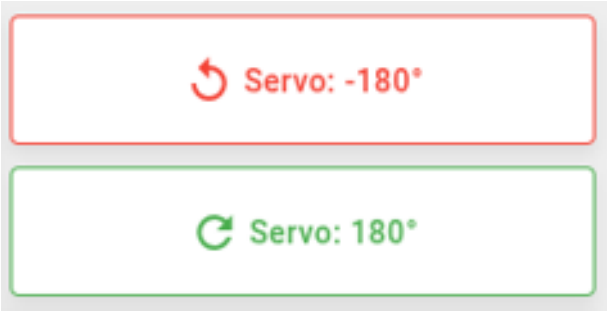


FIGURE 26 – Boutons permettant de contrôler le servomoteur depuis Thingsboard

4.5 Mise en place de TrustZone

La carte nRF7002 DK repose sur un SoC nRF5340 de Nordic Semiconductor, composé de deux cœurs Cortex-M33, chacun compatible avec la technologie Arm TrustZone. TrustZone permet de diviser l'exécution logicielle en deux mondes isolés :

- un monde sécurisé pour les fonctions sensibles (chiffrement, stockage sécurisé, contrôle d'accès),
- un monde non sécurisé pour le reste de l'application (acquisition de données, interface utilisateur, communication BLE...).

Dans l'environnement Zephyr, TrustZone est activé grâce à l'intégration de Trusted Firmware-M (TF-M). Celui-ci permet de définir des services sécurisés (Secure Services) auxquels le code non sécurisé peut accéder uniquement via des interfaces bien contrôlées (API sécurisées). Cette architecture garantit que même si le monde non sécurisé est compromis, les opérations critiques restent protégées.

Dans le cadre de notre projet, nous avons constaté que les données transmises via BLE (par exemple les valeurs de capteurs) étaient envoyées en clair, sans chiffrement ni authentification. Cela représente une faille de sécurité potentielle, car un attaquant pourrait intercepter ou manipuler ces données sensibles.

Pour répondre à cette problématique, nous avons décidé de s'appuyer sur TrustZone afin d'isoler les fonctions critiques de chiffrement dans le monde sécurisé. L'idée est de garantir que :

- le chiffrement des données avant leur envoi par BLE est effectué uniquement dans le monde sécurisé,
- les clés cryptographiques utilisées pour le chiffrement sont stockées et protégées contre tout accès non autorisé,
- l'application non sécurisée (celle qui gère le BLE) ne voit que les données déjà chiffrées, sans jamais manipuler directement les clés.

Concrètement, l'application non sécurisée collecte les valeurs de capteurs, puis fait appel à un service sécurisé exposé via TF-M pour obtenir une version chiffrée des données. Cette séparation permet de préserver la confidentialité des données, même en cas de vulnérabilité dans le code applicatif BLE.

Une fois les données chiffrées dans le monde sécurisé, elles sont transmises à l'application non sécurisée, qui se charge de les publier via le protocole MQTT vers la plateforme ThingsBoard. Du point de vue de l'application non sécurisée, le contenu de la donnée est opaque : seule sa version chiffrée est manipulée et transmise.

Cependant, la plateforme ThingsBoard doit pouvoir interpréter ces données pour les afficher dans un tableau de bord. Pour cela, une étape de déchiffrement côté serveur est nécessaire.

Deux approches sont possibles :

— soit ThingsBoard intègre la clé de déchiffrement, ce qui implique de lui faire confiance pour maintenir cette clé en sécurité ;

— soit un service intermédiaire (comme un script Python) connecté à ThingsBoard reçoit les messages MQTT, effectue le déchiffrement, puis retransmet les données en clair à la plateforme.

Cette chaîne garantit que les données restent chiffrées sur tout le trajet radio, depuis le microcontrôleur jusqu'au point de traitement, réduisant ainsi les risques d'interception ou de falsification. Le rôle de TrustZone ici est donc double : protéger la génération et l'usage des clés au niveau matériel, et restreindre strictement l'accès aux données sensibles à une zone d'exécution isolée.

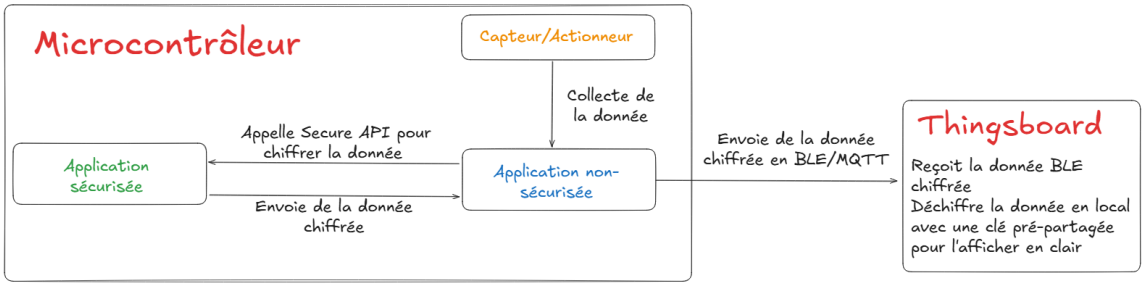


FIGURE 27 – Schéma de transmission sécurisée des données avec TrustZone et chiffrement BLE vers ThingsBoard

4.6 Tests et scénarios de cybersécurité

Dans un premier temps, nous avons réalisé des tests de cybersécurité afin d'observer le comportement par défaut des transmissions BLE effectuées depuis notre microcontrôleur nRF7002 DK vers la plateforme ThingsBoard. L'objectif était d'identifier si des données sensibles (issues de capteurs, comme la température ou l'humidité) pouvaient être interceptées en clair via une simple écoute du trafic.

Pour cela, nous avons utilisé un nRF52 configuré en mode sniffer BLE à l'aide de l'outil nRF Sniffer pour Wireshark. L'objectif était de simuler un attaquant passif capable d'intercepter les trames BLE échangées entre le nRF7002 et Thingsboard sans avoir besoin d'accéder physiquement au microcontrôleur.

La capture d'écran ci-dessous montre un paquet BLE de type Handle Value Notification transmis du périphérique (notre carte nRF7002) vers un central (ici un Raspberry Pi), contenant les données de température.

On observe clairement, dans la section Bluetooth Attribute Protocol, les éléments suivants :

— Le handle : 0x001e (identifiant caractéristique de température)

— La valeur de température : 27.25

Cela signifie que les données sont transmises sans chiffrement et qu'un attaquant passif (équipé d'un sniffer) peut visualiser directement la valeur de température en clair.

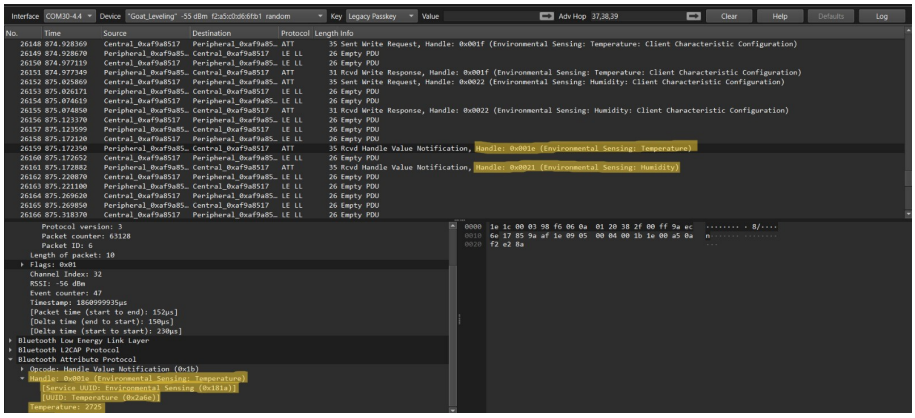


FIGURE 28 – Interception d'une donnée capteur non chiffrée

Ensuite, nous avons tenté d'effectuer une analyse similaire sur la transmission Wi-Fi, cette fois en utilisant un ESP32 comme sniffer pour capturer les paquets échangés. Contrairement au Bluetooth Low Energy, nous avons constaté que les données échangées par Wi-Fi étaient déjà chiffrées, et donc illisibles sans la clé de sécurité du réseau. Cela s'explique par le fait que le protocole Wi-Fi implémente un chiffrement natif (WPA2) au niveau de la couche liaison, ce qui empêche un sniffer classique comme notre ESP32 de lire les données en clair sans une authentification préalable sur le réseau.

Cette constatation renforce notre choix d'intégrer une solution de chiffrement logiciel (via TrustZone) pour les échanges BLE, car contrairement au Wi-Fi, la sécurité y est optionnelle.

Nous allons mettre en place une démonstration pratique d'une attaque de type Man-in-the-Middle (MITM), afin de mieux illustrer les vulnérabilités potentielles d'un système IoT non sécurisé.

Cette simulation aura pour but de montrer comment un attaquant peut intercepter ou manipuler les communications entre un appareil IoT et une plateforme distante comme ThingsBoard, en exploitant le manque de chiffrement applicatif.

Le réseau Wi-Fi utilisé pour la communication sera préconfiguré avec des informations connues (SSID, mot de passe), permettant à l'attaquant d'y accéder sans difficulté.

Nous utiliserons un second appareil pour simuler l'attaquant. Les types d'attaques envisagées sont :

- Injection de fausses données : une fois connecté au réseau, l'attaquant pourra usurper l'identité du capteur et envoyer de fausses valeurs à ThingsBoard.
- Déni de service (DoS) : l'attaquant pourra saturer le réseau ou la passerelle en envoyant un grand volume de messages, dans le but de bloquer la transmission des vraies données.
- Sniffing passif : en capturant les trames BLE ou MQTT, l'attaquant pourra analyser les paquets et récupérer des données.

L'intérêt de cette démonstration est de mettre en lumière l'importance du chiffrement des données, et justifie pleinement l'usage de technologies comme TrustZone pour protéger les opérations critiques (chiffrement, génération de clés...).

4.7 Conception du circuit et implantation des capteurs/actionneurs

Avant de débuter la phase de réalisation, nous avons reçu plusieurs indications techniques sur les contraintes à respecter pour la conception physique de la maquette. L'une des contraintes majeures concernait son gabarit global : la maquette devait être suffisamment compacte pour être transportable facilement, notamment en pouvant loger dans le coffre d'une voiture.

Après plusieurs échanges et estimations, nous avons retenu des dimensions maximales de 1,20 mètre de largeur, 1,40 mètre de longueur et 0,60 mètre de hauteur. Ces dimensions ont permis de garantir à la fois une surface de jeu suffisante pour rendre le fonctionnement lisible et dynamique, tout en respectant les exigences de transportabilité.

Une fois ces contraintes établies, nous avons entamé une phase de conception graphique du circuit. L'objectif principal était de définir un tracé fluide et fonctionnel, représentant le parcours de billes au sein de la maquette.

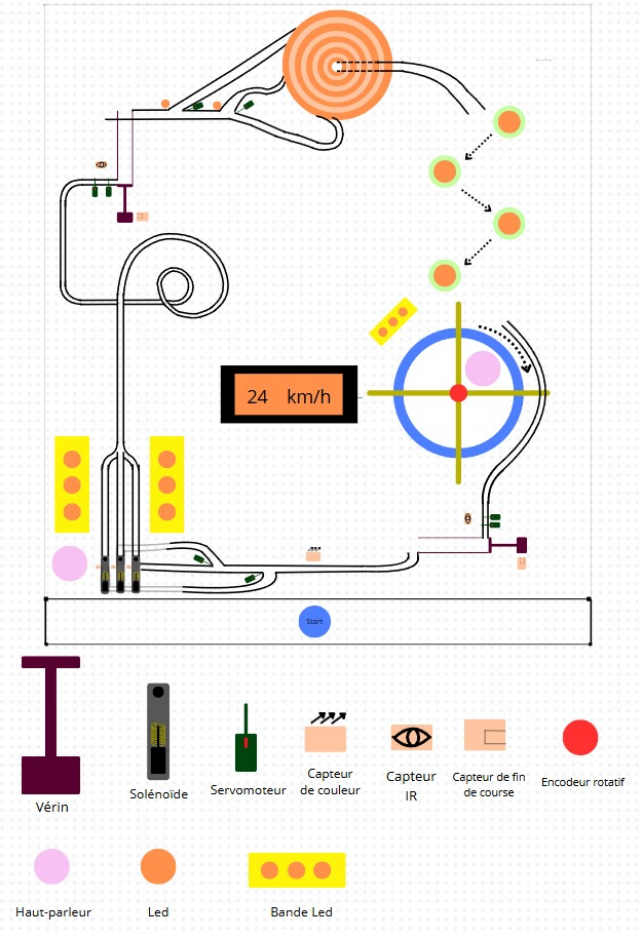


FIGURE 29 – Schéma du circuit

En parallèle de cette modélisation du circuit, nous avons également positionné les différents capteurs et actionneurs à des endroits clés du tracé, en tenant compte :

- des zones de détection nécessaires (capteur de couleur, fin de course, IR),
- des mécanismes de propulsion ou de manipulation (vérin, solénoïdes, servomoteurs),
- et des effets visuels et sonores (bandes LED, haut-parleur).

Ce travail de pré-positionnement nous a permis d'anticiper les besoins matériels, en listant les composants électroniques nécessaires, et de préparer les achats de capteurs et d'actionneurs adaptés à notre architecture.

Après avoir défini l'architecture du circuit et positionné les capteurs et actionneurs sur un schéma fonctionnel, nous sommes passés à l'étape de modélisation 3D. Pour cela, nous avons utilisé le logiciel Fusion 360, qui nous a permis de concevoir une maquette numérique fidèle à l'échelle réelle.

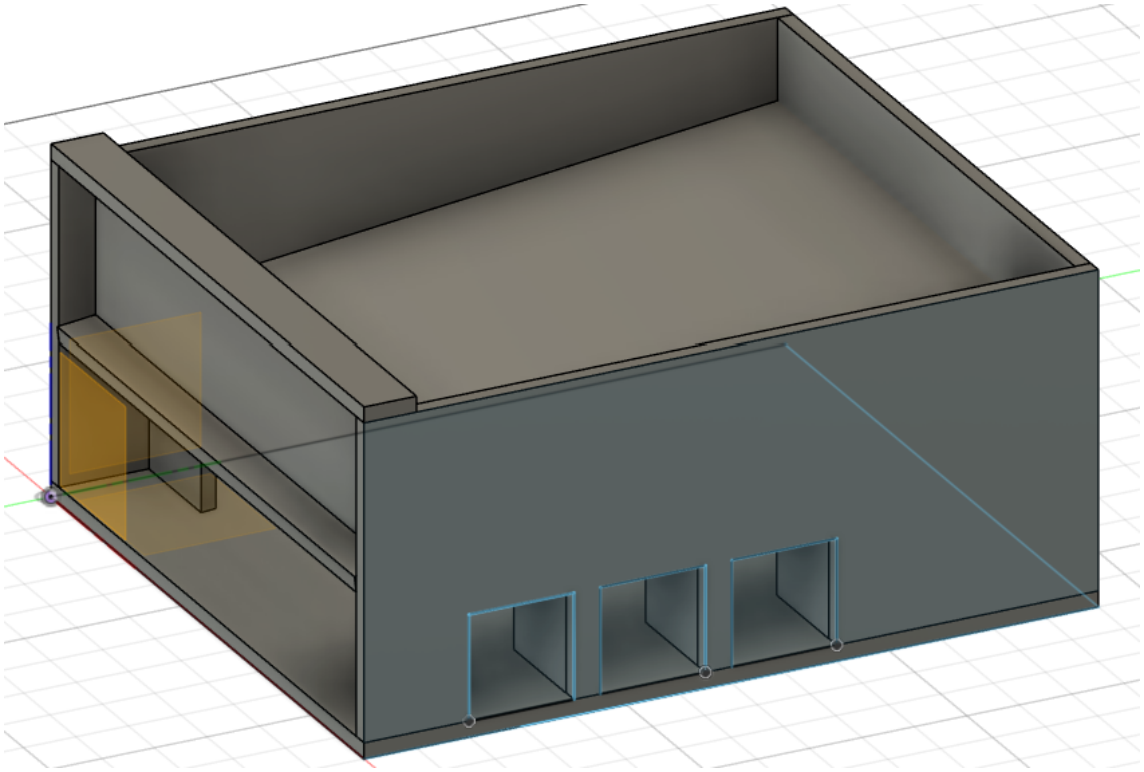


FIGURE 30 – Modélisation 3D de la maquette réalisée avec Fusion 360

4.8 Difficultés rencontrées

Au cours du projet, plusieurs obstacles ont été rencontrés, nécessitant parfois des ajustements techniques, des recherches approfondies ou des changements de stratégie.

L'un des premiers problèmes majeurs a concerné la mise en place du point d'accès Wi-Fi. Dans un premier temps, nous avons tenté une configuration manuelle, qui s'est révélée fastidieuse et source de nombreuses erreurs. Pour simplifier la gestion du réseau, nous avons ensuite opté pour NetworkManager, qui nous a permis une configuration plus claire et centralisée. Cependant, un autre obstacle est alors apparu : les microcontrôleurs n'arrivaient toujours pas à se connecter. Après plusieurs vérifications, nous avons découvert que le point d'accès utilisait un mode de sécurité non compatible. Il a donc fallu reconfigurer la sécurité en WPA2, seul protocole accepté par les microcontrôleurs, afin d'assurer la compatibilité et la stabilité de la connexion.

Par ailleurs, nous avons tenté de sniffer les données transitant en Wi-Fi via MQTT, dans le but d'observer le contenu des échanges comme nous l'avions fait pour le BLE. Malgré plusieurs tests et approches, nous avons conclu que cette opération était impossible dans notre contexte, les données étant soit chiffrées, donc inaccessible avec les outils à disposition.

Enfin, l'intégration de scénarios de cybersécurité a nécessité un apprentissage approfondi sur des notions complexes telles que la TrustZone, les protocoles sécurisés, ou encore les failles de communication sans fil. Une phase importante de documentation et d'expérimentation a donc été indispensable pour maîtriser suffisamment ces concepts et les adapter à un usage pédagogique.

La conception du circuit à billes a été sans doute la plus grande difficulté rencontrée au cours du projet. En effet, à ce stade, nous ne disposions d'aucun élément matériel concret : il a donc fallu imaginer l'ensemble de la maquette en s'appuyant uniquement sur des modélisations numériques. Ce manque de visibilité matérielle nous a plongés dans un flou important, rendant difficile la prise de décision. Nous ne savions pas exactement dans quelle direction aller, ni comment structurer efficacement les zones du circuit. Cela a freiné l'avancement initial, car chaque choix demandait d'anticiper les contraintes physiques et fonctionnelles sans pouvoir les tester. Ce n'est qu'après plusieurs croquis, discussions techniques et maquettes 3D que les grandes lignes ont pu être clarifiées.

5 Conclusion

5.1 Bilan général

Ce projet a marqué une étape importante dans la conception d'un système connecté complet, mêlant système embarqué, communication sans fil, supervision à distance et sécurité. La maquette, bien qu'encore en cours de développement, constitue déjà une base fonctionnelle solide. Toutefois, plusieurs axes d'amélioration restent envisageables afin de renforcer ses performances, sa fiabilité et sa sécurité.

Malgré les contraintes initiales et les nombreuses difficultés rencontrées notamment liées à la configuration réseau, à l'intégration matérielle ou à la cybersécurité nous avons su progresser en maintenant une dynamique constante grâce à une organisation méthodique, une collaboration étroite et une capacité d'adaptation.

Ce projet nous a permis de développer une compréhension concrète des défis rencontrés dans un système IoT multi-protocoles, ainsi que de prendre conscience de l'importance d'une approche structurée dès la phase de conception. L'intégration de scénarios de cybersécurité a également été une opportunité précieuse pour sensibiliser aux risques concrets, même dans un cadre pédagogique.

Enfin, bien que la maquette soit encore en phase de finalisation, les bases posées permettent d'envisager des perspectives d'évolution claires et réalistes. Ce travail nous a offert une expérience riche, tant sur le plan technique que sur le plan humain, et constitue une excellente préparation à la conduite de futurs projets professionnels plus ambitieux.

6 Résumé

6.1 Résumé du stage

Ce stage s'inscrit dans le cadre de la conception et du développement d'une maquette pédagogique connectée, intégrant plusieurs microcontrôleurs (STM32, nRF7002) et une grande diversité de capteurs et d'actionneurs. L'objectif était de réaliser un système IoT local, capable de transmettre des données via BLE, Wi-Fi et MQTT vers une interface de supervision ThingsBoard.

Durant ce stage, j'ai participé à toutes les étapes : modélisation du circuit, développement embarqué, intégration matérielle, configuration réseau, visualisation des données, mais aussi mise en place de scénarios de cybersécurité (sniffing, usurpation, injection). Ce projet m'a permis de développer des compétences en électronique, en communication sans fil, en cybersécurité et en gestion de projet.

La maquette est encore en cours de finalisation, mais les bases sont solides et les perspectives d'évolution nombreuses. Ce stage m'a offert une expérience technique concrète et enrichissante, dans un contexte de collaboration et d'autonomie.

6.2 Internship Summary

This internship focused on designing and developing a connected educational prototype involving multiple microcontrollers (STM32, nRF7002) and a wide range of sensors and actuators. The goal was to build a local IoT system capable of transmitting data via BLE, Wi-Fi, and MQTT to a supervision interface (ThingsBoard).

Throughout the internship, I was involved in every phase of the project : circuit modeling, embedded development, hardware integration, network configuration, data visualization, and even implementing cybersecurity scenarios (sniffing, spoofing, data injection). This experience allowed me to strengthen my skills in electronics, wireless communication, cybersecurity, and project management.

While the prototype is still under development, the system already provides a solid foundation with great potential for future improvements. This internship has been a highly valuable technical and collaborative experience.